Bac Général Maths + NSI (Mathématiques et Numérique et sciences informatiques)

CORRIGÉ NON OFFICIEL

Proposition de corrigé rédigée par FormaV avec l'aide d'outils d'intelligence artificielle. Elle peut contenir des erreurs et ne remplace ni le corrigé ni le barème officiels de l'examen.

Les sujets officiels des sessions passées et nos sujets d'entraînement sont classés sur https://bac-maths-nsi.fr/annales-sujets-et-corriges

Document mis en ligne par FormaV®. Reproduction ou diffusion sans autorisation interdite.

Numérique et sciences informatiques

Session 2026 • Antilles-Guyane • Jour 1

Code sujet **26-NSIJ1AG1** 3 heures 30

Proposition de corrigé pédagogique • FormaV

Se repérer dans le corrigé

Exercice 1 — Modèle relationnel, systèmes de gestion de bases de données relationnelles et langage SQL
12 réponses • sujet p. 2–4

Exercice 2 — Recherche d’une plus longue sous-liste commune et programmation Python
8 réponses • sujet p. 5–8

Exercice 3 — Protocoles de routage, sécurisation des communications et programmation orientée objet
16 réponses • sujet p. 9–13

Les numéros, parties et pages renvoient au sujet original. Chaque réponse rappelle la consigne, explique la démarche et donne le résultat.

FormaV - document d'entraînement non officiel

Exercice 1

6 points

Modèle relationnel, systèmes de gestion de bases de données relationnelles et langage SQL

Question 1.

Sujet p. 2

> Donner deux avantages que peut apporter un système de gestion de bases de données par rapport à l'utilisation d'un simple tableur dans lequel on ajouterait chaque nouvelle commande sur une ligne.

Un SGBDR permet d'organiser les données en plusieurs tables reliées : les informations sur un client ou une pizza ne sont enregistrées qu'une seule fois, puis les commandes les référencent par leurs identifiants. Cela limite les répétitions et les risques d'incohérence par rapport à un tableur contenant une ligne complète par commande.

Il permet également d'imposer des contraintes d'intégrité, par exemple l'unicité d'une clé primaire et la validité des références entre tables grâce aux clés étrangères.

> **RÉPONSE À RETENIR**
>
> Deux avantages sont :
>
> - la diminution des redondances et des incohérences de données ;
> - le contrôle de l'intégrité des données grâce aux clés primaires et étrangères.

Question 2.

Sujet p. 2, 3

> Indiquer, en justifiant, s'il est possible d'avoir deux pizzas avec le même numéro de pizza, le même prix et avec une couleur de pizza différente, à partir du schéma relationnel fourni dont les clés primaires sont soulignées.

Dans la relation `pizza`, l'attribut `num` est souligné dans le schéma : c'est la clé primaire. Une clé primaire identifie de manière unique chaque ligne d'une relation ; sa valeur ne peut donc pas être partagée par deux pizzas distinctes.

> **RÉPONSE À RETENIR**
>
> **Non.** Deux pizzas ne peuvent pas avoir le même numéro, même si elles ont le même prix : `num` est la clé primaire de `pizza`. Il ne peut donc pas exister deux lignes de même numéro et de couleurs différentes.

Question 3.

Sujet p. 2, 3

> Donner un exemple de clé étrangère en précisant la relation à laquelle elle appartient ainsi que l'attribut et la relation qu'elle référence.

Une commande concerne un client. L'attribut `id_client` de la relation `commande` contient l'identifiant du client ayant passé cette commande ; il doit correspondre à un identifiant présent dans la relation `client`.

> **RÉPONSE À RETENIR**
>
> Un exemple est : `commande.id_client` est une clé étrangère de la relation `commande` qui référence l'attribut `id` de la relation `client`.

FormaV - document d'entraînement non officiel

Question 4.

Sujet p. 2, 3

> Écrire le résultat renvoyé par la requête SQL suivante, en s'appuyant uniquement sur les extraits des tables fournis précédemment : SELECT prix FROM pizza WHERE couleur = 'Bianca';

Dans l'extrait de la table `pizza`, les deux pizzas dont la couleur est `Bianca` sont les pizzas 2 et 4. Leurs prix sont respectivement 11.5 et 15.5. La requête ne sélectionne que l'attribut `prix`.

RÉPONSE À RETENIR

prix
11.5
15.5

Question 5.

Sujet p. 3

> Écrire une requête SQL qui donne les couleurs des pizzas dont le prix est strictement supérieur à 15 €.

Il faut sélectionner dans `pizza` les lignes dont le prix est strictement supérieur à 15, puis ne renvoyer que leur couleur.

```
SELECT couleur
FROM pizza
WHERE prix > 15;
```

Dans l'extrait fourni, seule la pizza numéro 4, de prix 15.5, satisfait cette condition.

RÉPONSE À RETENIR

couleur
Bianca

Question 6.

Sujet p. 2, 3, 4

> Écrire une requête SQL qui donne les prénoms des clients qui n'ont pas payé une pizza qui leur a pourtant été livrée.

Les commandes recherchées vérifient simultanément `livraison = 1` et `paiement = 0`. Une jointure avec `client` permet alors d'obtenir le prénom. `DISTINCT` évite de répéter le prénom si un même client possède plusieurs commandes répondant au critère.

```
SELECT DISTINCT client.prenom
FROM client
JOIN commande ON client.id = commande.id_client
WHERE commande.livraison = 1
  AND commande.paiement = 0;
```

Dans l'extrait, seule la commande 42364 répond à ces conditions. Elle concerne le client 106, Raffaele Panzini.

FormaV - document d'entraînement non officiel

RÉPONSE À RETENIR

prenom
Raffaele

Question 7.

Sujet p. 2, 3, 4

Écrire une requête SQL qui donne le prix moyen de vente des pizzas commandées par un client de nom « Esposito ».

Il faut relier `client` à `commande` afin de sélectionner les commandes de clients nommés Esposito, puis relier `commande` à `pizza` pour accéder au prix de chaque pizza commandée. La fonction `AVG` calcule la moyenne de ces prix.

```
SELECT AVG(pizza.prix)
FROM client
JOIN commande ON client.id = commande.id_client
JOIN pizza ON commande.num_pizza = pizza.num
WHERE client.nom = 'Esposito';
```

Dans l'extrait, Giulia Esposito a commandé les pizzas 2, 3 et 1, aux prix respectifs 11.5, 12.5 et 11.5. Ainsi :

$$\frac{11,5+12,5+11,5}{3}=\frac{35,5}{3}=11,833333\ldots$$

RÉPONSE À RETENIR

AVG(pizza.prix)
11.833333…

Question 8.

Sujet p. 3, 4

La commande de code 42365 vient tout juste d'être livrée. Écrire une requête SQL qui permet de mettre à jour la base de données en conséquence.

La commande 42365 vient d'être livrée : il faut donc modifier seulement son attribut `livraison`, qui passe de 0 à 1. L'énoncé n'indique aucun changement concernant le paiement, qui reste donc inchangé.

```
UPDATE commande
SET livraison = 1
WHERE code = 42365;
```

RÉPONSE À RETENIR

La ligne de commande de code 42365 est mise à jour avec `livraison = 1`.

FormaV - document d'entraînement non officiel

Question 9.

Sujet p. 2, 3, 4

> Le 3 janvier 2025, Emanuele Girasole, qui n'a jamais encore commandé dans ce restaurant, souhaite se faire livrer la pizza numéro 2. Écrire, dans le bon ordre, les requêtes SQL qui permettent de mettre à jour la base de données en conséquence. On suppose que l'identifiant client 107 et que le code de commande 42366 n'ont pas encore été attribués dans la base de données.

Le nouveau client doit être créé avant sa commande, car la commande référence son identifiant. Il souhaite une livraison, mais celle-ci n'a pas encore été effectuée au moment de l'enregistrement : conformément à la définition de l'attribut, on prend donc `livraison = 0`. Aucun règlement n'est mentionné ; on adopte la convention explicite `paiement = 0`.

```
INSERT INTO client (id, nom, prenom)
VALUES (107, 'Girasole', 'Emanuele');

INSERT INTO commande (code, id_client, num_pizza, date, livraison, paiement)
VALUES (42366, 107, 2, '2025-01-03', 0, 0);
```

RÉPONSE À RETENIR

Il faut d'abord insérer Emanuele Girasole comme client d'identifiant 107, puis insérer la commande 42366, qui référence ce client et la pizza 2. La commande est initialement non livrée.

Question 10.

Sujet p. 2, 3, 4

> La gérante décide de retirer de la vente la pizza d'identifiant 1 et exécute la requête DELETE FROM pizza WHERE num = 1;. Indiquer le problème que peut poser l'exécution de cette requête.

La pizza d'identifiant 1 apparaît encore dans la table `commande` : les commandes 42364 et 42365 ont la valeur `num_pizza = 1`.

Si `commande.num_pizza` est une clé étrangère référençant `pizza.num`, un SGBDR respectant l'intégrité référentielle doit refuser la suppression. Sans cette contrainte, les commandes concernées feraient référence à une pizza inexistante.

RÉPONSE À RETENIR

Cette suppression peut violer l'intégrité référentielle, car des commandes existantes référencent toujours la pizza 1.

Question 11.

Sujet p. 2, 4

> Écrire le schéma relationnel des deux nouvelles tables ingredient et composition, afin de retrouver les ingrédients utilisés pour chaque pizza, sachant que chaque pizza comporte divers ingrédients et qu'un même ingrédient peut être utilisé sur plusieurs pizzas. Indiquer clairement les clés primaires et les clés étrangères.

Une pizza peut contenir plusieurs ingrédients et un ingrédient peut être utilisé dans plusieurs pizzas : il s'agit d'une association plusieurs-à-plusieurs. La relation `composition` est donc une table d'association entre `pizza` et `ingredient`.

On choisit ici un identifiant d'ingrédient nommé `id_ingredient`. Dans `composition`, le couple `(num_pizza, id_ingredient)` est une clé primaire composée : il identifie une composition et empêche de déclarer deux fois le même ingrédient pour la même pizza.

FormaV - document d'entraînement non officiel

Relation	Clé primaire	Clé(s) étrangère(s)
ingredient	id_ingredient	Aucune
composition	(num_pizza, id_ingredient)	num_pizza vers pizza(num) id_ingredient vers ingredient(id_ingredient)

RÉPONSE À RETENIR

`ingredient(`**`id_ingredient`** `: entier, nom : texte)`

`composition(`**`num_pizza`** `: entier,` **`id_ingredient`** `: entier)`

Dans `composition`, `num_pizza` est à la fois une composante de la clé primaire et une clé étrangère vers `pizza(num)` ; `id_ingredient` est à la fois une composante de la clé primaire et une clé étrangère vers `ingredient(id_ingredient)`.

Question 12.

Sujet p. 4

> Afin de permettre à chaque client de consulter sur Internet l'historique de ses commandes, la base de données serait placée sur un serveur accessible par des requêtes extérieures et non uniquement via le réseau interne du restaurant. Indiquer les problèmes soulevés par cette démarche et les précautions à prendre pour y remédier.

Rendre le service accessible depuis Internet augmente la surface d'attaque. Des personnes non autorisées pourraient chercher à consulter l'historique d'un autre client, usurper un compte, modifier ou supprimer des données, intercepter des échanges, réaliser une injection SQL ou rendre le service indisponible. Les données de commandes sont des données personnelles et doivent être protégées.

Les précautions comprennent notamment :

- **Confidentialité :** utiliser HTTPS/TLS afin de chiffrer les échanges entre le navigateur et le serveur ; ne pas exposer directement le SGBDR sur Internet, mais passer par une application web ou une API sécurisée.
- **Authentification et autorisation :** mettre en place des comptes clients avec une authentification robuste et vérifier côté serveur qu'un client authentifié ne peut accéder qu'à son propre historique ; attribuer à chaque compte de base de données les droits minimaux nécessaires.
- **Intégrité :** employer des requêtes paramétrées pour prévenir les injections SQL, valider les données reçues et conserver les contraintes de la base.
- **Disponibilité et suivi :** maintenir les logiciels à jour, filtrer les accès avec un pare-feu, journaliser les accès et prévoir des sauvegardes régulières dont la restauration est testée.

RÉPONSE À RETENIR

L'accès extérieur crée des risques de fuite de données, d'accès frauduleux, d'altération et d'indisponibilité. Il faut notamment chiffrer les communications, authentifier les clients, contrôler précisément leurs droits, protéger l'application des injections, maintenir les systèmes à jour et prévoir pare-feu, journaux et sauvegardes.

Exercice 2

6 points

Recherche d'une plus longue sous-liste commune et programmation Python

Partie A.

Question 1.

Sujet p. 5, 6

> Recopier et compléter le tableau « Évolution des paramètres lors des appels récursifs », en indiquant la valeur des paramètres lors des appels récursifs, pour un appel initial sous_listes([9, 3, 7], [[]]).

À chaque appel non terminal, `liste.pop()` retire le dernier élément de `liste`. On ajoute alors à l'accumulateur les sous-listes obtenues en plaçant cet élément devant chacune des sous-listes déjà présentes.

Ainsi, on retire successivement `7`, puis `3`, puis `9`.

RÉPONSE À RETENIR

Appel	liste	accumulateur
appel initial	`[9, 3, 7]`	`[[]]`
1er sous-appel	`[9, 3]`	`[[], [7]]`
2nd sous-appel	`[9]`	`[[], [7], [3], [3, 7]]`
dernier sous-appel	`[]`	`[[], [7], [3], [3, 7], [9], [9, 7], [9, 3], [9, 3, 7]]`

Lors du dernier sous-appel, `liste` est vide : la condition de la ligne 2 est satisfaite et cet accumulateur est renvoyé.

Question 2.

Sujet p. 5, 6

> Recopier et compléter les lignes 7 et 8 du code de la fonction sous_listes : tmp.append(...) puis return sous_listes(...).

Pour chaque `sous_liste` de l'accumulateur, la liste temporaire reçoit la sous-liste formée de `element`, placé en tête, suivi de `sous_liste`. L'appel récursif reçoit ensuite les anciennes sous-listes et les nouvelles.

RÉPONSE À RETENIR

```
        tmp.append([element] + sous_liste)
    return sous_listes(liste, accumulateur + tmp)
```


FormaV - document d'entraînement non officiel

Question 3.

Sujet p. 6, 7

> Expliquer pourquoi l'exécution de la fonction lcs_force_brute risque de prendre beaucoup de temps selon la longueur des listes passées en paramètre. Justifier.

Si `liste_1` contient *m* éléments, chaque élément peut être soit retenu, soit non retenu dans une sous-liste. La fonction `sous_listes` construit donc 2^m sous-listes, en comptant les choix de positions ; certaines listes ainsi construites peuvent avoir la même valeur si la liste contient des répétitions.

La fonction `lcs_force_brute` parcourt toutes ces sous-listes. Pour chacune, `est_extraite` peut parcourir les *n* éléments de `liste_2`. Le nombre d'opérations croît donc de façon exponentielle avec *m*, au plus de l'ordre de $O(2^m(m+n))$ si l'on tient compte de la construction des sous-listes et de leurs tests.

RÉPONSE À RETENIR

La recherche exhaustive devient rapidement très lente : augmenter *m* d'une unité double approximativement le nombre de sous-listes à examiner.

Partie B.

Question 4.

Sujet p. 7

> Indiquer ce que renvoie l'appel suivant : lcs([9, 3, 7, 5, 8], 1, [9, 7, 8, 3, 7, 3], 1).

Les paramètres `1` et `1` indiquent que l'on ne considère que les premiers éléments des deux listes, c'est-à-dire `[9]` et `[9]`. Ces deux listes ont `9` comme unique élément commun.

RÉPONSE À RETENIR

```
lcs([9, 3, 7, 5, 8], 1, [9, 7, 8, 3, 7, 3], 1)
```

renvoie `[9]`.

Question 5.

Sujet p. 7

> On suppose que L_1 et L_2 sont des listes non vides dont les derniers éléments sont identiques : m ≥ 1, n ≥ 1, L_1[m-1] et L_2[n-1] égaux. Écrire la relation de récurrence pour obtenir une plus longue sous-liste commune aux deux listes L_1 et L_2 à partir du résultat de lcs(L_1, m-1, L_2, n-1).

Les derniers éléments des préfixes considérés sont égaux. Une plus longue sous-liste commune aux préfixes de tailles `m - 1` et `n - 1` peut donc être prolongée par cet élément commun, qui est à la fois `L_1[m-1]` et `L_2[n-1]`.

RÉPONSE À RETENIR

La relation de récurrence est :

$$\mathrm{lcs}(L_1, m, L_2, n) = \mathrm{lcs}(L_1, m-1, L_2, n-1) + [L_1[m-1]].$$

La concaténation conserve l'ordre dans les deux listes et ajoute un élément à la sous-liste commune.

FormaV - document d'entraînement non officiel

Question 6.

Sujet p. 8

> Justifier que la fonction aux contenue dans la fonction lcs_top_down est récursive.

Une fonction est récursive lorsqu'elle s'appelle elle-même. C'est le cas de `aux` :

- ligne 6, `aux` est appelée avec `index_x - 1` et `index_y - 1` lorsque les deux éléments sont égaux ;
- lignes 7 et 8, `aux` est appelée avec l'un des deux indices diminué de 1 lorsque les éléments sont différents.

Ces appels portent sur des préfixes plus petits. Ils finissent par atteindre le cas de base des lignes 3 et 4 : si l'un des indices est négatif, la fonction renvoie `[]`.

> **RÉPONSE À RETENIR**
>
> `aux` est donc une fonction récursive, car elle s'appelle elle-même jusqu'au cas de base où un indice devient négatif.

Question 7.

Sujet p. 8

> Détailler le risque d'un tel choix dans la présente situation, lorsque les listes L_1 et L_2 sont très grandes.

Pour de très grandes listes, une branche d'appels peut avoir une profondeur de l'ordre de $m + n$: à chaque appel, un indice au moins diminue, avant que l'un d'eux devienne négatif. Or Python limite la profondeur de sa pile d'appels. L'exécution risque donc de s'interrompre avec une erreur `RecursionError` avant d'atteindre le cas de base.

En outre, lorsque les éléments comparés sont différents, les lignes 7 et 8 effectuent deux appels récursifs. En l'absence de mémorisation, un même sous-problème, identifié par un même couple d'indices, peut être recalculé de nombreuses fois. Le temps de calcul peut alors aussi devenir très grand.

> **RÉPONSE À RETENIR**
>
> La récursivité expose donc la fonction à un dépassement de la pile d'appels pour de grandes listes ; l'absence de mémoïsation entraîne aussi des recalculs très nombreux.

Question 8.

Sujet p. 8

> Proposer une solution pour éviter le problème évoqué à la question 7.

On peut remplacer les appels récursifs par une programmation dynamique itérative. On construit un tableau `longueurs` dans lequel `longueurs[i][j]` contient la longueur d'une LCS des `i` premiers éléments de `liste_x` et des `j` premiers éléments de `liste_y`.

Les deux boucles calculent chaque sous-problème une seule fois. La reconstruction finale diminue strictement `i` ou `j` à chaque étape : elle termine. Cette solution ne crée aucun appel récursif. Une simple mémoïsation supprimerait les recalculs, mais ne supprimerait pas à elle seule le risque de profondeur de pile excessive.

RÉPONSE À RETENIR

```
def lcs_iterative(liste_x: list, liste_y: list) -> list:
    n = len(liste_x)
    m = len(liste_y)
    longueurs = [[0] * (m + 1) for _ in range(n + 1)]

    for i in range(1, n + 1):
        for j in range(1, m + 1):
            if liste_x[i - 1] == liste_y[j - 1]:
                longueurs[i][j] = longueurs[i - 1][j - 1] + 1
            else:
                longueurs[i][j] = max(longueurs[i - 1][j],
                                      longueurs[i][j - 1])

    resultat = []
    i, j = n, m
    while i > 0 and j > 0:
        if liste_x[i - 1] == liste_y[j - 1]:
            resultat.append(liste_x[i - 1])
            i -= 1
            j -= 1
        elif longueurs[i - 1][j] >= longueurs[i][j - 1]:
            i -= 1
        else:
            j -= 1

    resultat.reverse()
    return resultat
```

Avec les listes de l'énoncé, la reconstruction suit notamment la trace suivante.

Couple `(i, j)`	Décision	Élément ajouté
`(5, 6)`, puis `(4, 6)`	Les éléments diffèrent ; les égalités de longueurs conduisent à diminuer `i`.	aucun
`(3, 6)`	Les éléments diffèrent ; on diminue `j`.	aucun
`(3, 5)`	Les deux éléments valent `7`.	`7`
`(2, 4)`	Les deux éléments valent `3`.	`3`
`(1, 3)`, puis `(1, 2)`	Les éléments diffèrent ; on diminue deux fois `j`.	aucun
`(1, 1)`	Les deux éléments valent `9`.	`9`

```
lcs_iterative([9, 3, 7, 5, 8], [9, 7, 8, 3, 7, 3])
```

renvoie `[9, 3, 7]`, qui est bien une LCS. La construction du tableau effectue $O(mn)$ calculs et utilise $O(mn)$ mémoire.

Exercice 3

8 points

Protocoles de routage, sécurisation des communications et programmation orientée objet

Partie A

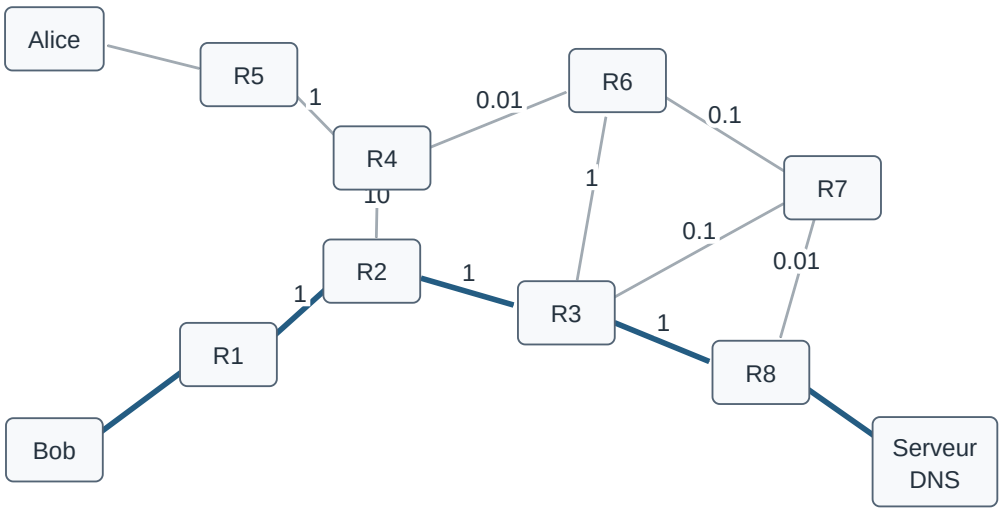
Question 1.

Sujet p. 9, 10

> Donner, en utilisant le protocole RIP, le chemin que les paquets emprunteront afin qu'une requête de Bob parvienne au serveur DNS. Ne faire apparaître que les routeurs traversés.

Le protocole RIP choisit le chemin contenant le plus petit nombre de routeurs, sans tenir compte des coûts inscrits sur les liaisons. Pour atteindre le serveur DNS, le chemin passant par R1, R2, R3 puis R8 traverse quatre routeurs ; les autres chemins en traversent davantage.

Réseau : chemin RIP de Bob vers le serveur DNS



Les arêtes mises en évidence forment le chemin qui minimise le nombre de routeurs traversés.

> **RÉPONSE À RETENIR**
>
> Routeurs traversés : **R1** → **R2** → **R3** → **R8**.

Question 2.

Sujet p. 9, 10

> On suppose que Bob a l'adresse IP d'Alice. Donner, en utilisant le protocole OSPF, le chemin que les paquets emprunteront afin qu'une requête de Bob parvienne à Alice. Ne faire apparaître que les routeurs traversés.

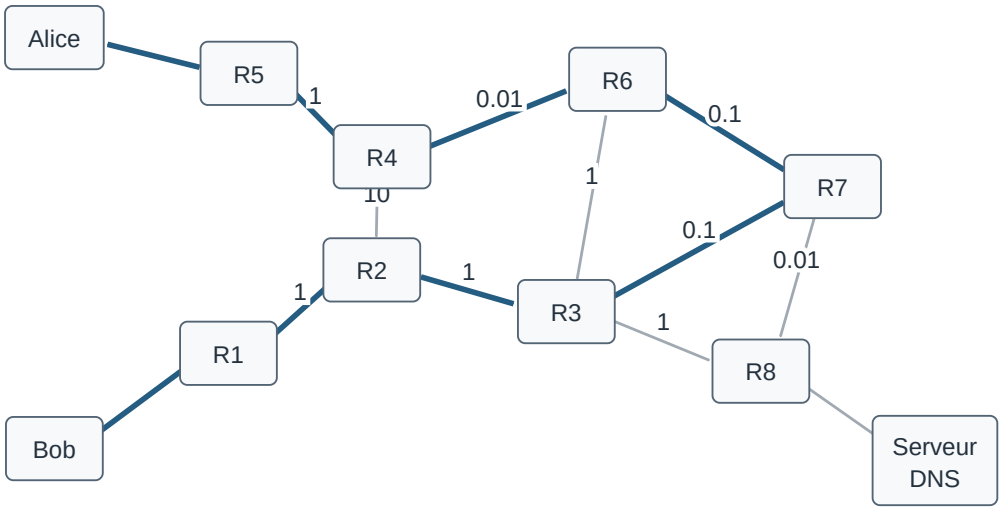
OSPF minimise cette fois la somme des coûts. Depuis R2, la liaison directe vers R4 coûte 10. Le chemin R2 → R3 → R7 → R6 → R4 coûte seulement :

$1 + 0{,}1 + 0{,}1 + 0{,}01 = 1{,}21.$

Il est donc préférable, puis il reste à passer par R5 pour atteindre Alice.

FormaV - document d'entraînement non officiel

Réseau : chemin OSPF de Bob vers Alice



Les arêtes mises en évidence constituent le chemin de coût minimal vers Alice.

> **RÉPONSE À RETENIR**
>
> Routeurs traversés : **R1** → **R2** → **R3** → **R7** → **R6** → **R4** → **R5**.

Question 3.

Sujet p. 9, 10

> Donner le coût du chemin identifié dans la question 2.

On additionne les coûts des liaisons du chemin de la question 2. Les liaisons Bob–R1 et R5–Alice ne portent aucun coût indiqué sur la figure.

$$1 + 1 + 0{,}1 + 0{,}1 + 0{,}01 + 1 = 3{,}21.$$

> **RÉPONSE À RETENIR**
>
> Le coût du chemin OSPF est **3,21**.

Question 4.

Sujet p. 9, 10

> Expliquer ce qui se passerait si le routeur R3 tombe en panne. L'écriture des tables de routage n'est pas attendue.

Si R3 tombe en panne, il ne peut plus recevoir ni transmettre de paquets. Les routes qui passent par R3 deviennent donc indisponibles. Après détection de la panne, les routeurs mettent à jour leurs informations de routage et calculent des chemins ne passant pas par R3.

Par exemple, Alice reste accessible depuis Bob par R1 → R2 → R4 → R5. Le serveur DNS reste accessible par R1 → R2 → R4 → R6 → R7 → R8. Une interruption temporaire est possible pendant la mise à jour des routes.

RÉPONSE À RETENIR

Les paquets sont **redirigés par des chemins de remplacement évitant R3**, après convergence du routage.

Question 5.

Sujet p. 10

> Indiquer un protocole permettant à un navigateur et un serveur web de communiquer en chiffrant leurs données.

Le protocole HTTPS permet à un navigateur et à un serveur web de communiquer en chiffrant les données ; HTTPS repose sur TLS.

RÉPONSE À RETENIR

Protocole attendu : **HTTPS**.

Question 6.

Sujet p. 10

> Bob génère une clé symétrique qu'il doit transmettre à Alice ; Bob et Alice possèdent alors cette même clé symétrique. Expliquer le rôle qu'aura cette clé symétrique dans leur communication.

La clé symétrique est une clé secrète commune à Bob et à Alice. Elle sert ensuite à chiffrer les données de leurs échanges et à les déchiffrer : l'émetteur chiffre avec cette clé et le destinataire retrouve les données avec la même clé.

Elle assure donc la confidentialité des échanges, à condition qu'elle reste secrète. À elle seule, elle ne permet pas de prouver l'identité de l'émetteur.

RÉPONSE À RETENIR

La clé symétrique sert à **chiffrer et déchiffrer les données échangées** entre Bob et Alice.

Question 7.

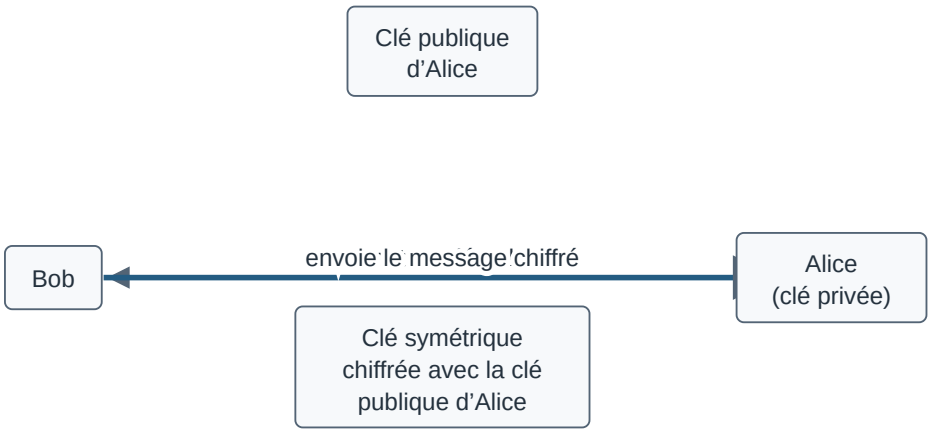
Sujet p. 10

> Alice génère un couple de clés composé d'une clé publique et d'une clé privée. Expliquer comment Alice et Bob vont utiliser cette clé publique et cette clé privée afin que Bob puisse transmettre de façon sécurisée sa clé symétrique à Alice. Ne pas parler, dans cette question, de signature ni de certificat.

Alice transmet sa clé publique à Bob. Bob chiffre la clé symétrique qu'il a créée avec cette clé publique, puis il envoie le résultat chiffré à Alice. Alice utilise sa clé privée, qui doit rester secrète, pour déchiffrer le message et récupérer la clé symétrique.

Un tiers peut connaître la clé publique d'Alice, mais il ne peut pas déchiffrer le message sans la clé privée correspondante. Cette confidentialité suppose que Bob a bien reçu l'authentique clé publique d'Alice, et non une clé substituée par un attaquant.

Transmission confidentielle de la clé symétrique



Seule la détentrice de la clé privée d'Alice peut déchiffrer la clé symétrique.

> **RÉPONSE À RETENIR**
>
> **Alice → Bob :** clé publique d'Alice ; **Bob → Alice :** clé symétrique chiffrée avec cette clé publique ; **Alice :** déchiffrement avec sa clé privée.

Question 8.

Sujet p. 10, 11

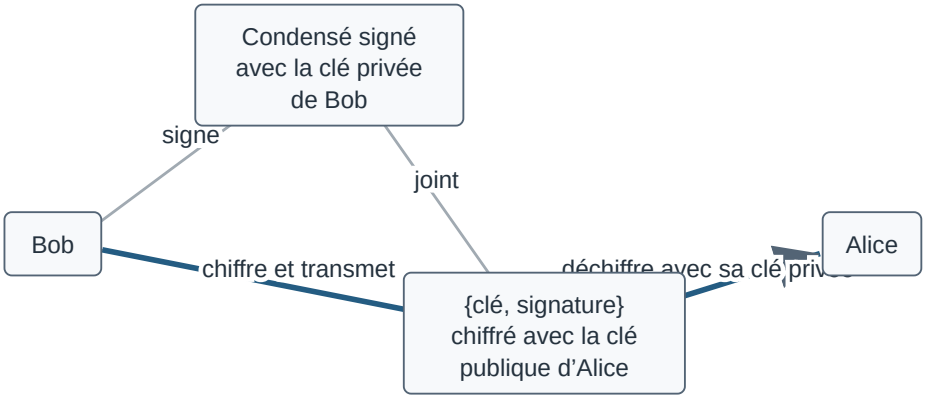
> Bob envoie sa clé publique à Alice, crée un condensé des données de la clé symétrique, signe ce condensé avec sa clé privée, puis transmet à Alice l'ensemble composé de la clé symétrique et de la signature après l'avoir chiffré grâce à la clé publique transmise par Alice. Expliquer pourquoi Alice est la seule à pouvoir accéder à la clé symétrique et peut avoir la certitude que c'est bien Bob qui la lui a envoyée.

Bob chiffre l'ensemble formé de la clé symétrique et de la signature avec la clé publique d'Alice. Seule Alice détient la clé privée associée : elle seule peut donc déchiffrer cet ensemble et accéder à la clé symétrique. Cela apporte la **confidentialité**.

Alice déchiffre ensuite la signature grâce à la clé publique de Bob, ce qui lui fournit le condensé que Bob avait signé. Elle calcule elle-même le condensé de la clé symétrique reçue et compare les deux condensés. S'ils sont identiques, la clé reçue n'a pas été altérée : son **intégrité** est contrôlée. Une signature valide ne pouvant être produite qu'avec la clé privée de Bob, elle établit aussi l'**authenticité** de l'émetteur.

Cette dernière conclusion suppose que la clé publique utilisée est bien celle de Bob, donc que le lien entre cette clé et Bob est digne de confiance.

Confidentialité et authentification de la clé



Alice vérifie la signature avec la clé publique fiable de Bob puis compare les condensés.

> **RÉPONSE À RETENIR**
>
> La clé privée d'Alice garantit que seule Alice lit la clé symétrique ; la signature vérifiée avec la clé publique fiable de Bob garantit **l'intégrité** de la clé et son **origine Bob**.

Partie B

Question 9.

Sujet p. 11

> Écrire le code d'une fonction somme qui prend en paramètre un tableau, de type list en Python, d'entiers et renvoie la somme des éléments de ce tableau. Exemple : somme([1, 3, 5, 7, 9]) renvoie 25.

La variable `s` est un accumulateur. Après le traitement de chaque élément, elle contient la somme des éléments déjà parcourus ; après la boucle, elle contient donc la somme complète du tableau.

```
def somme(tab):
    s = 0
    for element in tab:
        s = s + element
    return s
```

Trace pour `somme([1, 3, 5, 7, 9])` :

Élément traité	Valeur de s
départ	0
1	1
3	4
5	9
7	16
9	25

RÉPONSE À RETENIR

`somme([1, 3, 5, 7, 9])` renvoie **25**.

Question 10.

Sujet p. 11

Écrire le code d'une fonction permuter qui prend en paramètres un tableau tab, de type list en Python, et deux entiers i et j, puis modifie le tableau tab en permutant ses éléments d'indices i et j.

Il faut mémoriser temporairement la valeur située à l'indice `i`, afin qu'elle ne soit pas perdue quand cette case est remplacée par la valeur d'indice `j`.

```
def permuter(tab, i, j):
    temporaire = tab[i]
    tab[i] = tab[j]
    tab[j] = temporaire
```

Avec `tab = [1, 3, 5, 7, 9]`, `i = 1` et `j = 3`, on conserve 3, on place 7 à l'indice 1, puis on place 3 à l'indice 3.

RÉPONSE À RETENIR

Après `permuter(liste, 1, 3)`, `liste` vaut **[1, 7, 5, 3, 9]**.

Question 11.

Sujet p. 11

Recopier et compléter la ligne 3 du code de la fonction inverser qui modifie le tableau passé en paramètre en inversant l'ordre de ses éléments : def inverser(tab): for i in range(len(tab) // 2): permuter(...).

À l'itération d'indice `i`, l'élément d'indice `i` doit être échangé avec l'élément symétrique, d'indice `len(tab) - 1 - i`. La boucle ne parcourt que la première moitié du tableau afin de ne pas échanger une même paire deux fois.

```
def inverser(tab):
    for i in range(len(tab) // 2):
        permuter(tab, i, len(tab) - 1 - i)
```

RÉPONSE À RETENIR

Ligne 3 : `permuter(tab, i, len(tab) - 1 - i)`.

Question 12.

Sujet p. 11, 12

> Recopier et compléter les lignes 7 à 13 du code de la fonction generer_sac. Le sac est un tableau de 8 entiers : son premier élément est choisi aléatoirement entre 1 et 5 inclus ; chaque nouvel entier est choisi aléatoirement entre S+1 et 2*S inclus, où S est la somme des éléments déjà présents ; une fois les huit entiers obtenus, l'ordre des éléments est inversé afin que le premier élément soit le plus grand.

Le sac contient d'abord un entier entre 1 et 5. La boucle est exécutée sept fois, ce qui ajoute sept entiers et donne bien huit entiers au total. Après chaque calcul de `s`, le nouvel entier est choisi dans l'intervalle imposé. Comme chaque nouvel entier est strictement supérieur à la somme précédente, il est plus grand que tous les entiers déjà présents ; son inversion finale place donc le plus grand au début.

```
from random import randint

def generer_sac():
    '''
    renvoie un sac (8 entiers)
    : return (list) un sac
    '''
    sac = [randint(1, 5)]
    for i in range(7):
        s = somme(sac)
        nouvel_entier = randint(s + 1, 2 * s)
        sac.append(nouvel_entier)
    inverser(sac)
    return sac
```

Exemple de trace possible avant l'inversion : `[1, 2, 6, 10, 32, 71, 205, 568]`.

Somme S avant ajout	Intervalle autorisé	Valeur choisie
1	[2 ; 2]	2
3	[4 ; 6]	6
9	[10 ; 18]	10
19	[20 ; 38]	32
51	[52 ; 102]	71
122	[123 ; 244]	205
327	[328 ; 654]	568

RÉPONSE À RETENIR

Avec cet exemple, la fonction renvoie après inversion **[568, 205, 71, 32, 10, 6, 2, 1]**.

Partie C

Question 13.

Sujet p. 12, 13

> Recopier et compléter la ligne 4 du code de la classe Cle_symetrique, initialisée avec un unique attribut sac, tableau renvoyé par la fonction generer_sac du module outils : import outils ; class Cle_symetrique() ; def __init__(self): ... = ...

L'attribut d'instance est `sac`. Il reçoit le tableau créé par la fonction `generer_sac` du module `outils`.

```
import outils

class Cle_symetrique():
    def __init__(self):
        self.sac = outils.generer_sac()
```

RÉPONSE À RETENIR

Ligne 4 : `self.sac = outils.generer_sac()`.

Question 14.

Sujet p. 12, 13

> Recopier et compléter les lignes 8 à 11 du code de la méthode chiffrer, qui multiplie chaque bit d'indice i de l'octet par l'entier d'indice i du sac de l'instance, puis renvoie la somme de tous les produits obtenus.

L'octet contient huit bits, d'indices 0 à 7. Pour chaque indice, le programme multiplie le bit par l'élément de même indice du sac, puis additionne les huit produits.

```
def chiffrer(self, octet):
    '''
    renvoie le chiffrement de octet
    : paramètre octet (list) un tableau de huit 0 ou 1
    : return (int)
    '''
    tab_produits = []
    for i in range(8):
        produit = octet[i] * self.sac[i]
        tab_produits.append(produit)
    s = outils.somme(tab_produits)
    return s
```

Pour le sac de l'énoncé et l'octet `[1, 0, 1, 1, 1, 1, 0, 1]`, le tableau des produits est `[568, 0, 71, 32, 10, 6, 0, 1]`, de somme 688.

RÉPONSE À RETENIR

Lignes complétées : `range(8)`, `octet[i] * self.sac[i]` et `outils.somme(tab_produits)`.

Question 15.

Sujet p. 12, 13

> Justifier qu'avec la méthode dechiffrer décrite, le déchiffrement du nombre 688 est [1, 0, 1, 1, 1, 1, 0, 1].

On applique l'algorithme de déchiffrement au sac `[568, 205, 71, 32, 10, 6, 2, 1]`. Le reste initial vaut 688.

Indice i	Valeur du sac	Reste avant test	Bit	Reste après test
0	568	688	1	120
1	205	120	0	120
2	71	120	1	49
3	32	49	1	17
4	10	17	1	7
5	6	7	1	1
6	2	1	0	1
7	1	1	1	0

RÉPONSE À RETENIR

Les bits successivement obtenus sont **[1, 0, 1, 1, 1, 1, 0, 1]**.

Question 16.

Sujet p. 13

> Écrire le code de la méthode dechiffrer de la classe Cle_symetrique qui prend en paramètre un entier à déchiffrer et renvoie un tableau représentant l'octet correspondant à cet entier.

Le sac est parcouru du plus grand poids au plus petit. Si le reste est au moins égal au poids courant, le bit vaut 1 et ce poids est retiré ; sinon, le bit vaut 0. Il y a exactement huit itérations, une par bit de l'octet.

```
def dechiffrer(self, entier):
    octet = []
    for i in range(8):
        if entier >= self.sac[i]:
            octet.append(1)
            entier = entier - self.sac[i]
        else:
            octet.append(0)
    return octet
```

La construction du sac garantit qu'avant l'inversion chaque nouvel entier est strictement supérieur à la somme de tous les précédents. Après inversion, chaque poids est donc supérieur à la somme des poids qui le suivent : le choix glouton permet ainsi de retrouver sans ambiguïté les bits d'un entier produit par `chiffrer`.

RÉPONSE À RETENIR

Pour le sac de l'exemple, `ma_cle.dechiffrer(688)` renvoie **[1, 0, 1, 1, 1, 1, 0, 1]**.

Sujet : https://www.sujetdebac.fr/annales-pdf/2026/spe-numerique-informatique-2026-antilles-guyane-1-sujet-officiel.pdf
© FormaV EI • Proposition de correction, distincte d'un corrigé officiel.

Copyright © 2027 FormaV®. Tous droits réservés.

FormaV® est une marque déposée auprès de l'INPI sous le numéro 5227897.

Ce document a été élaboré par FormaV® avec le plus grand soin afin d'accompagner chaque apprenant vers la réussite de ses examens. Son contenu (textes, graphiques, méthodologies, tableaux, exercices, concepts, mises en forme) constitue une œuvre protégée par le droit d'auteur.

Toute copie, partage, reproduction, diffusion ou mise à disposition, même partielle, gratuite ou payante, est strictement interdite sans accord préalable et écrit de FormaV®, conformément aux articles L.111-1 et suivants du Code de la propriété intellectuelle. Dans une logique anti-plagiat, FormaV® se réserve le droit de vérifier toute utilisation illicite, y compris sur les plateformes en ligne ou sites tiers.

En utilisant ce document, vous vous engagez à respecter ces règles et à préserver l'intégrité du travail fourni. La consultation de ce document est strictement personnelle.

Merci de respecter le travail accompli afin de permettre la création continue de ressources pédagogiques fiables et accessibles.